High Speed Motion Estimation Using Stream Processing

Somayeh Sardashti
Department of Electrical and Computer Eng.
University of Tehran
s.sardashti@ece.ut.ac.it

Omid Fatemi
Department of Electrical and Computer Eng.
University of Tehran
omid@fatemi.net

Abstract: Media processing applications are dominant in many new workloads today. These applications including 3D graphics, image compression, and signal processing, require tens to hundreds of billions of computations per second. To achieve real-time requirements, current media processors use special purpose architectures. Because of the changes in media applications and standards, flexibility is an important factor in new media processors. Hence, programmable media processors are required. Recently, stream processing has been shown to be a suitable candidate for programmable media processors. In this paper, we propose an implementation of motion estimation algorithm onto Imagine, Stanford stream processor. Simulation results show the speed-up of up to 20 compared to sequential processors.

Keywords: Stream Processor, Imagine Processor, SIMD, Motion Estimation.

1 Introduction
Media processing applications demand large amount of operations because of their real time nature. To perform media processing in real-time, current media processors use special-purpose architectures for common specific application. Such processors require significant design effort and are thus difficult to change as media-processing applications and algorithms evolve. Programmable Stream architectures like Imagine exploit the locality and concurrency existing in media applications by partitioning the communication and storage structures to support many ALUs efficiently [1] [2]. The paper is organized as follows. The characteristics of media processing applications are presented in section 2. Stream processor is introduced in section 3. Imagine processor is introduced in section 4, and our implementation of motion estimation application is explained followed by simulation results and conclusions in section 5.

2 Media Applications
Understanding the behavior of multimedia applications is crucial for determining the design space for hardware support. Media applications have three common basic characteristics.
1. Operations on one data element are almost always independent of operations on others. This characteristic results in a large amount of data parallelism.
2. There is little global data reuse, resulting in locality.
3. The applications are computationally intensive, resulting in 100 to 200 arithmetic operations per each element read from off-chip memory.
Hence, a suitable architecture exploits the concurrency and locality existing in media application. Stream processors are shown to be good candidates for extracting existing parallelism which are explained in the next section.

3 Stream Processors
Stream processors are digital signal processors (DSPs) targeted at high-performance embedded applications. They contain clusters of functional units and provide a bandwidth hierarchy, supporting hundreds of arithmetic units. They exploit instruction level parallelism (ILP) and sub-
word parallelism (SP) within a cluster and data parallelism (DP) across clusters.

The central idea behind stream processing is to organize an application into streams and kernels to expose the existing locality and concurrency in media processing applications [1] [2].

- **Streams**
  Streams contain a set of elements of the same type. Stream elements can be simple, such as a single number, or complex, such as the coordinates of a triangle in 3D space. Computational machines access elements of a stream as sequential records.

- **Kernels**
  Kernels are the computational units used on streams. They can have one or more input and output streams and perform complex calculations ranging from a few to thousands of operations per input element.
  The advantage of thinking in terms of kernels is that their global accesses are made explicit. For example, all data for the kernel in Imagine must be placed as an input or as an output stream in a kernel argument.

- **Full Applications**
  A full application is composed of multiple streams and kernels.
  Full search motion estimation is shown in Kernel-Stream manner in Figure 2.

4 **Imagine Processor**
Imagine is a programmable stream processor developed at Stanford University that is used for media applications. It consists of a programming model, software tools, and architecture, all designed to operate on streams [1] [2] [3].

4.1 **Imagine Architecture**
Imagine exploits the locality and concurrency existing in media applications by partitioning the data communication in the processor into three levels:
- Operands for arithmetic operations are kept locally in Local Register Files (LRFs) near the ALUs.
- Streams of data are stored in Stream Register File (SRF), which can transfer data to and from LRFs.
- Global data is stored on off-chip memory.

These three levels of storage form a data bandwidth hierarchy. Each level provides successively more storage and less bandwidth. Using this hierarchy, stream architecture can support many ALUs in an area and power efficient manner.

Imagine has a host processor which controls the operations by sending stream instructions. The main stream instructions are:
- **Load** transfers streams from off-chip DRAM to the SRF.
- **Store** transfers streams from the SRF to off-chip DRAM.
- **Receive** transfers streams from the network to the SRF.
- **Send** transfers streams from the SRF to the network.
- **Cluster op** executes a kernel in the arithmetic clusters that reads inputs streams from the SRF, computes output streams, and writes the output streams to the SRF.
- **Load microcode** loads streams consisting of kernel microcode—576-bit very long instruction word (VLIW) instructions from the SRF into the microcontroller instruction store (a total of 2,048 instructions).

Imagine architecture is shown in Figure 1. The processor consists of a 128-Kbyte stream register file (SRF), 48 floating-point arithmetic units organized in 8 arithmetic clusters which are controlled by a micro-controller in SIMD fashion, a network interface, a streaming memory system and a stream controller.

Each arithmetic cluster consists of eight functional units. Small LRFs are connected to the inputs of each arithmetic unit. An intra-cluster switch connects outputs of functional units to the inputs of LRFs. Clusters work on a Single Instruction Multiple Data (SIMD) manner. During kernel execution, one VLIW instruction is sent to all of them.

Kernels typically perform a compound stream operation on each element. A compound stream operation reads an element from its input stream(s) in the SRF and computes a series of arithmetic operations, and then it appends the results to output streams. It stores all temporary data in the LRFs, and final results on SRF.
4.2 Stream Programming model

The stream programming model allows simple control, makes communication explicit, and exposes the existing parallelism of media applications. A stream program organizes data as streams and expresses all computation as kernels. A stream is a sequence of similar elements. Each stream element is a record. A kernel consumes a set of input streams, performs a computation, and produces a set of output streams.

Programming Languages of StreamC and KernelC are used for modeling stream programs [4].

5 Motion Estimation on Imagine

Motion estimation has played an important role in video compression. Temporal correlation is being used in motion estimation algorithms for inter-frame video compression. One popular technique for motion estimation, as recommended in all MPEG and H.26x standards, is a block matching algorithm (BMA). In BMA, input frame is divided into non-overlapping blocks, and the blocks in previous frame are being searched for the best candidate block, based on a given matching criterion.

Full search algorithm gives the optimal result by searching through all possible blocks in the searching window; however, its high computational cost limits its practical use.

In this paper, we propose a new implementation of full search block matching algorithm on Imagine. In this algorithm, total absolute difference is used as criteria for choosing the best matching block.

Full search algorithm is implemented using Imagine programming tool set. Imagine Simulator (ISIM) is employed to obtain simulation results. Using ISIM, IPC (Instructions per Cycle), SRF (Stream Register File) BW (Bandwidth), and LRF (Local Register File) BW can be obtained. Results are shown in table 1.

<table>
<thead>
<tr>
<th>ALU</th>
<th>LRF BW (GB/s)</th>
<th>SRF BW (GB/s)</th>
<th>IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6 GOPS</td>
<td>98</td>
<td>0.08</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 1 Results of Full Search Algorithm of 16-bit 8-by-8 pixels macroblocks

The high IPC value indicates that the large arithmetic intensity and parallelism in this algorithm can be exploited by Imagine compilers. The register bandwidth data shows that more than
95% of data accesses are to cluster LRFs, demonstrating the large amount of locality within kernel computations. It shows that for every access to SRF, reading some stream records, processor performs a lot of calculations.

6 Conclusions

Streaming processors such as Imagine is suitable for media applications which demand a large scale of computations. In this paper, a computational intensive application, i.e. motion estimation, has been implemented on Imagine. The results show that this implementation is suitable for real-time motion estimation.

References


